```
import 'dart:io';
void main() {
  List<BankAccount> accounts = [];
  accounts.add(NormalAccount('001', 'Alice', 1000.0));
  accounts.add(SavingsAccount('002', 'Bob', 2000.0, 5.0));
bool countine = true;
while (countine) {
    print('chooice number');
    print('1. deposit');
    print('2. withdraw');
    print('3. check_balance');
    print('4. transfer');
    print('5. display balance');
    print('6. exit');
 String? choice = stdin.readLineSync();
switch (choice) {
      case '1':
        print('enter account number');
        String? accNumber = stdin.readLineSync();
        BankAccount? account = accounts.firstWhere((acc) => acc.accountNumber == accNumber,);
         if (account != null) {
          print('enter amount of deposit');
          double amount = double.parse(stdin.readLineSync() ?? '0');
          account.deposit(amount);
          print('deposit done');
        } else {
          print('number account not correct');
        }
        break;
 case '2':
        print('enter account number');
        String? accNumber = stdin.readLineSync();
        BankAccount? account = accounts.firstWhere((acc) => acc.accountNumber == accNumber, )
        if (account != null) {
          print('enter anoumt of withdraw');
          double amount = double.parse(stdin.readLineSync() ?? '0');
          account.withdraw(amount);
          print('withdraw done');
        } else {
          print('account number is not correct');
        }
        break;
      case '3':
        print('enter account number');
        String? accNumber = stdin.readLineSync();
        BankAccount? account = accounts.firstWhere((acc) => acc.accountNumber == accNumber,);
        if (account != null) {
          print('current balance\$${account.checkBalance().toStringAsFixed(2)}');
        } else {
          print('account number is not correct');
        }
        Break;
      case '4':
        print('enter source account number');
        String? sourceAccNumber = stdin.readLineSync();
        BankAccount? sourceAccount = accounts.firstWhere((acc) => acc.accountNumber ==
sourceAccNumber,);
         if (sourceAccount != null) {
          print('enter target account number');
          String? targetAccNumber = stdin.readLineSync();
          BankAccount? targetAccount = accounts.firstWhere((acc) => acc.accountNumber ==
targetAccNumber, );
            if (targetAccount != null) {
            print('enter amount of transfer');
            double amount = double.parse(stdin.readLineSync() ?? '0');
            sourceAccount.transfer(targetAccount, amount);
          } else {
            print('account number not correct');
          }
        } else {
          print('account number not correct');
        }
        break;

      case '5':
        for (var acc in accounts) {
          print(acc);
        }
        break;
    case '6':
       countine  = false;
        print('program end');
        break;
     default:
        print('choice correct number');
        break;
    }
  }
```

```
}

class BankAccount {
  String accountNumber;
  String name;
  double balance;
//constractor
  BankAccount(this.accountNumber, this.name, this.balance);
  void deposit(double amount) {
    if (amount > 0) {
      balance += amount;
    } else {
      print('amount of deposit must be positive');
    }
  }
  void withdraw(double amount) {
    if (amount > 0 && amount <= balance) {
      balance -= amount;
    } else {
      print('the amount of deposit is not diysple');
    }
  }
  double checkBalance() {
    return balance;
  }
  void transfer(BankAccount otherAccount, double amount) {
    if (amount > 0 && amount <= balance) {
      withdraw(amount);
      otherAccount.deposit(amount);
      print('transfer done \$${amount.toStringAsFixed(2)} to account ${otherAccount.name}.');
    } else {
      print('the amount of deposit is not diysple');
    }
  }
}

class NormalAccount extends BankAccount {
  NormalAccount(String accountNumber, String name, double balance)
      : super(accountNumber, name, balance);
}
class SavingsAccount extends BankAccount {
  double interestRate;

  SavingsAccount(String accountNumber, String accountHolder, double balance, this.interestRate)
```

```
      : super(accountNumber, accountHolder, balance);
 @override
  void deposit(double amount) {
    super.deposit(amount);
    double interest = balance * interestRate / 100;
    balance += interest;
  }
}
```

---------------------------------------topic search--

soild principles are a set of guidelines for writing clean, maintainable, and scalable object-oriented code.